CLO API/ SDK Guide

SDK Version
Applicable CLO S/W Version: CLO 7.1.194 Official Release and later, 23th December 2022
API/SDK v4.1.0

Document Version
Version 27 (2022/12/23)

Table of Contents

1. Introduction ..... **오류! 책갈피가 정의되어 있지 않습니다.**
2. Installation 3
3. CLO SDK Package 4
4. Quick Start 7
5. Make your Own Plug-in 9
6. Plug-In Manager 10
7. Library Window Implementation 12
8. Plug-in Menu Position 19
9. Plug-in Debug Logs 20

1. Introduction

This document describes how to download, install and use CLO API/SDK and how developers can implement their own plug-ins running on CLO. It also includes the simple structure for API/SDK package and addresses for the sample projects in the package.

As CLO S/W has been developed for cross-platforms: Windows and Mac OS, you can use the API/SDK package to make plug-in .dll and/or .dylib for both platforms.

They have commonalities but some parts are different. Will describe the common parts and different parts respectively.

2. Installation

1) System Environment

   a. Windows

      - OS: Windows 8, Windows 10
      - IDE: Visual Studio 2017 (or above)

   b. Mac OS

      - OS: mac OSX 10.12 (or above)
      - IDE: Xcode 9.2 (or above)

2) Download SDK

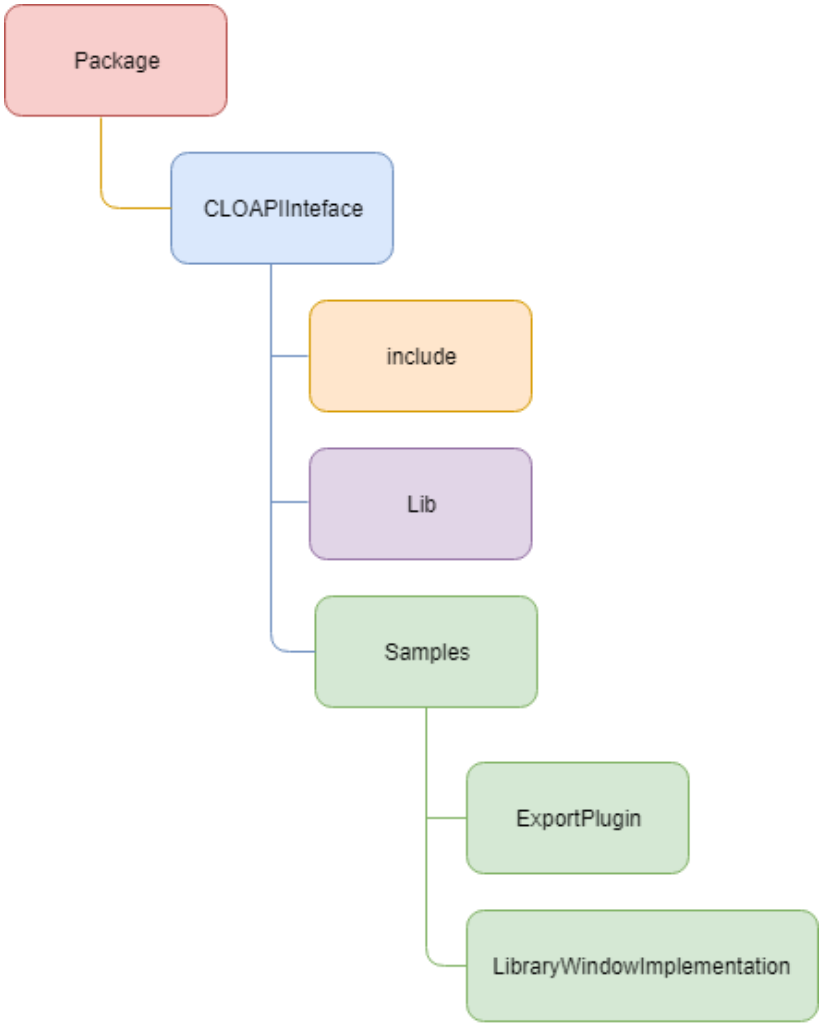
   Please download the SDK zip file from the online manual.

3. CLO SDK Package

CLO SDK package includes API interface header/lib files and sample projects.

1) Folder structure

- API/SDK Package



2) API/SDK Package

a. CLOAPIInterface folder

i. <api_sdk_package>

i) CLOAPIInterface.h

This file includes all the header files for interface classes located in the ‘include’ folder. You can include this file to use API calls inside the interface classes. The usage is described in the Samples -> ExportPlugin.

ii) LibraryWindowInterface.h

This file includes an interface class so that the plug-in developers can override to implement the Library Window construction. You can find the sample code for the usage via API -> Samples -> LibraryApiImplementation

ii. include

i) CloApiData.h
This file includes some structures/classes for API calls.

ii) DefineDllForWin.h

This file contains a ‘define’ for export/import dll.

iii) ExportAPIInterface.h, ImportAPIInterface.h

You can include this header file to import/export files such as ZPrj, ZPac, OBJ, Rendering images, and so on. You can find the example in “ExportPlugin” sample.

iv) FabricAPIInterface.h

You can include this header file to import metadata to add fabric and/or export zfab file.

v) RestAPIInterface.h

This includes high-level wrapping functions of REST APIs. If these are not enough for your own use, you may use your own REST API functions.

vi) UtilityAPIInteface.h

This file includes some utility functions like “Get temporary folder path of CLO” and “Show a message box on CLO”.

iii. Lib

This folder has library files for function table in the CLOAPIInterface project: CLOAPIInterface.lib for Windows or libCLOAPIInterface_.dylib for Mac OS. You should import this library file into your plug-in project to run in CLO API functions inside CLO S/W.

iv. Samples

i. ExportPlugin
A sample project to show how to use Export APIs and create a plug-in. For example, you can see how to save files such as thumbnail and Tech Pack and send them to your server using REST APIs with this project file.

ii. LibraryWindowImplementation
A sample project to show how to implement the custom Library Window to build up the Finder API tab. You can find the usage from **7. Library Window Implementation**

4. Quick Start

1) Download and install the “CLO 7.1.194 or later” into your PC via the CLO Official Site (https://www.clo3d.com).

2) Open the sample project – ExportPlugin project

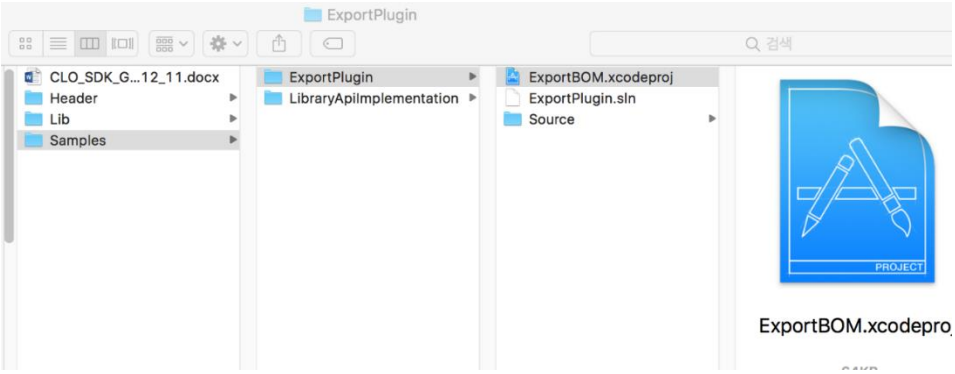
   a. Windows

      i. Build “ExportPlugin.dll” file

         - Open the solution file (ExportPlugin.sln) via Visual Studio 2017 (or above).
         - Run “Build Solution” in Visual Studio (press Ctrl+Shit+B for shortcut). Make sure that “Solution Configurations” is “Release” and “Solution Platforms” is “x64” when building the solution.
         - The DLL file will be created in “Samples\ExportPlugin\x64\Release” folder.

      ii. Put “ExportPlugin.dll” into the default plug-in folder.

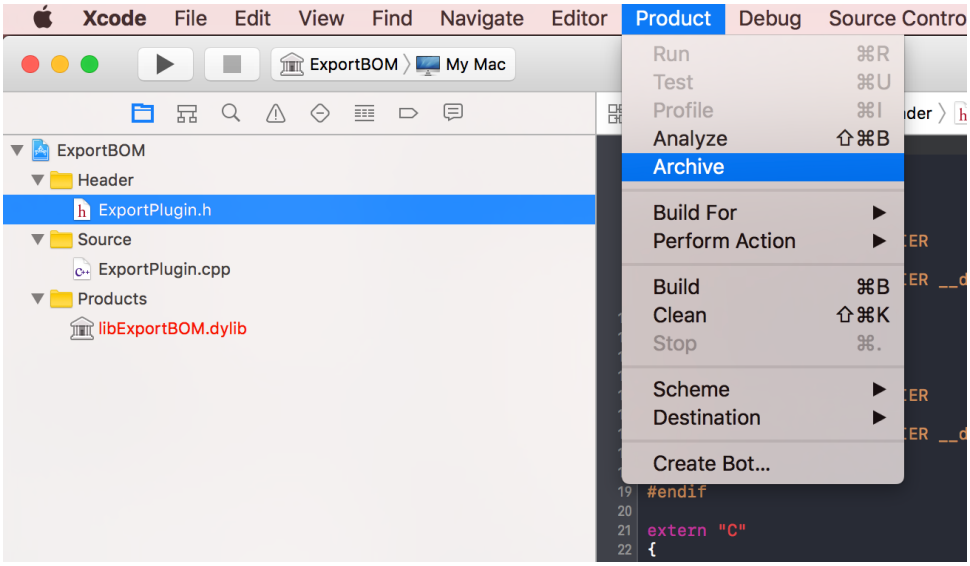
         - Copy the plug-in dll file and paste/overwrite into the assets folder; located in “C:\Users\Public\Documents\CLO\Assets\Preference-s\API_Plug_in\”
         - You can use the ‘cloapi_plugins’ folder as you used in the beta version of API/SDK packages. Create ‘cloapi_plugins’ into the CLO Executable folder you installed in i) and use it as the default plug-in folder.



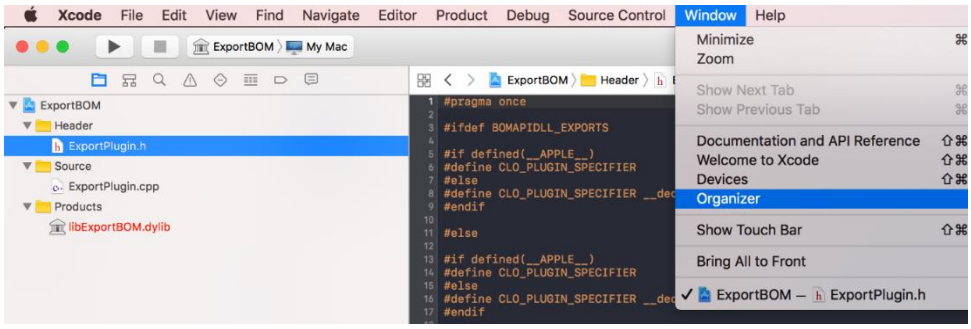
   b. Mac OS

      i. Build “libExportBOM.dylib” file

         - Open the xcode project file (ExportPlugin.xcodeproj) via Xcode.



- Run “Archive” via Product menu in XCode.

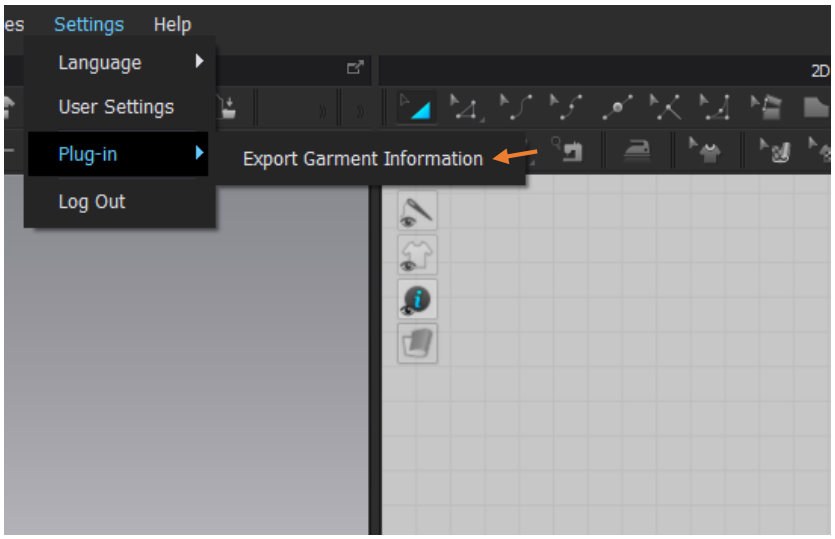


- You’d be able to find the output dylib via Window -> Organizer.

ii. Put “libExportBOM.dylib” into the default plug-in folder.

- Copy the plug-in dll file and paste/overwrite into the assets folder; located in “~Documents\clo\Assets\Preferences\API_Plug_in\”

3) Run CLO and browse the new feature from plug-in menu.



5. Make your Own Plug-in

To make your own plug-in, you need to customize the following functions. You can find these functions inside ‘ExportPlugin’ project.

1) DoFunction

   This function is called when a user clicks the action menu in the CLO S/W plug-in menu. In this function you can implement codes like sending exported files to your server.

2) GetActionName

   You can change the action menu title which appears under the plug-in menu.

3) CallbackFromWebKit(int argc, char** argv)

   You can write the code for the call back function which will be called after ‘Browser Window’ is triggered by Java Script code when using Web API. You will be able to get the number of messages from ‘argc’ as integer and each message from ‘argv’ as char* array via Web API sent from Java Script code on your Webpage.

4) GetObjectNameTreeToAddAction

   This function is called when the user adds a plug-in dll(or dylib) file into the CLO S/W. You can manage the position where you want to put the plug-in menu above or below which menu/action in the application. See “8. Plug-in menu position” for details.

5) GetPositionIndexToAddAction

   This function is also used when the user imports a plug-in dll(or dylib); you can choose whether the plug-in action would be put into below the designated menu/action which you wrote in 4) or above the target.

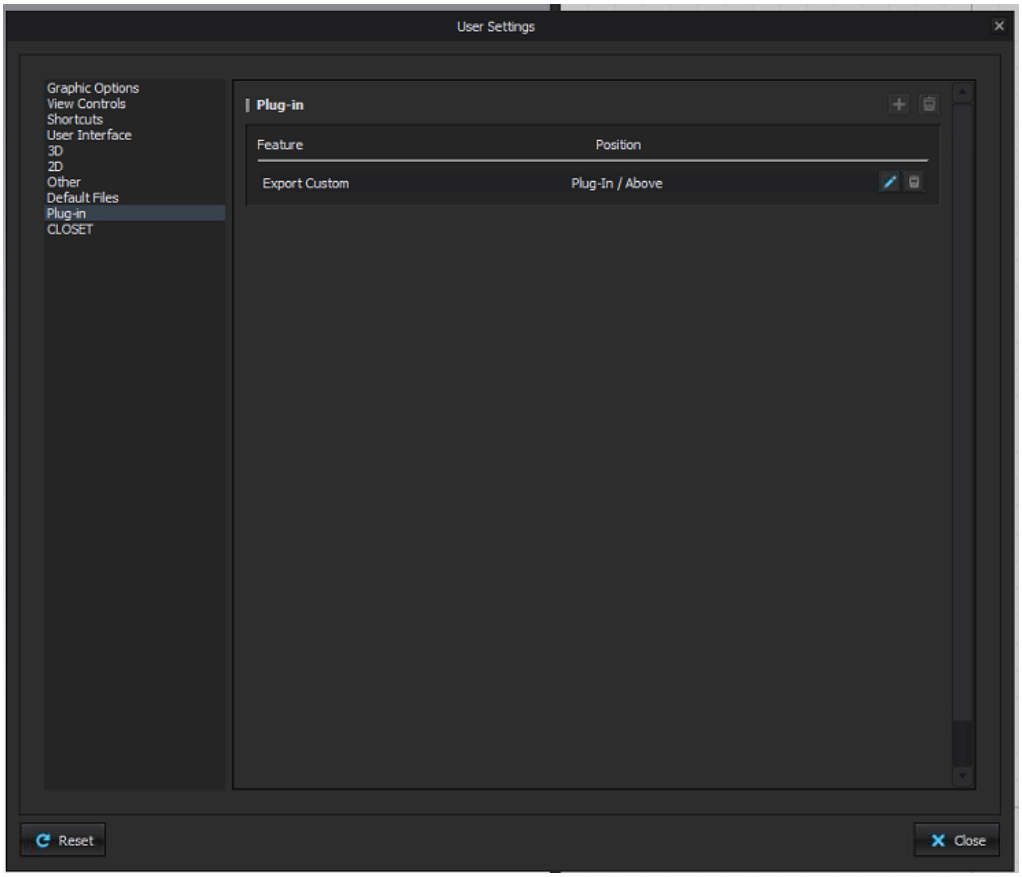
Then, build DLL(or dylib) and paste it to the installation folder as guided in “4. Quick Start”.

※ **Note**. If you rename the plugin file and put the file on the designated folder, CLO will run the plug-in just after completing the start-up of the application.

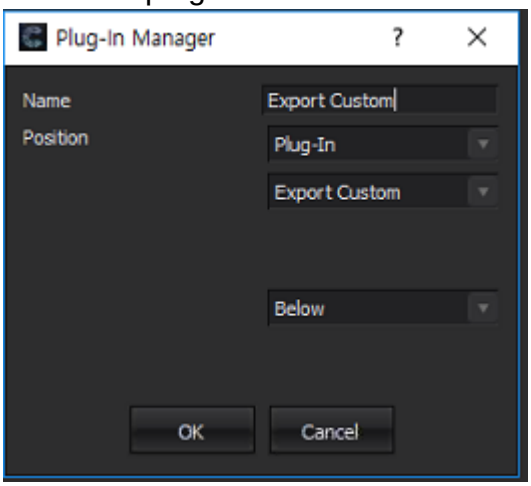
- Windows : “StartupPlugin.dll” into the executable folder,
- MacOS: “libStartupPlugin1.0.0.dylib” into the framework folder in the app package.

6. Plug-In Manager

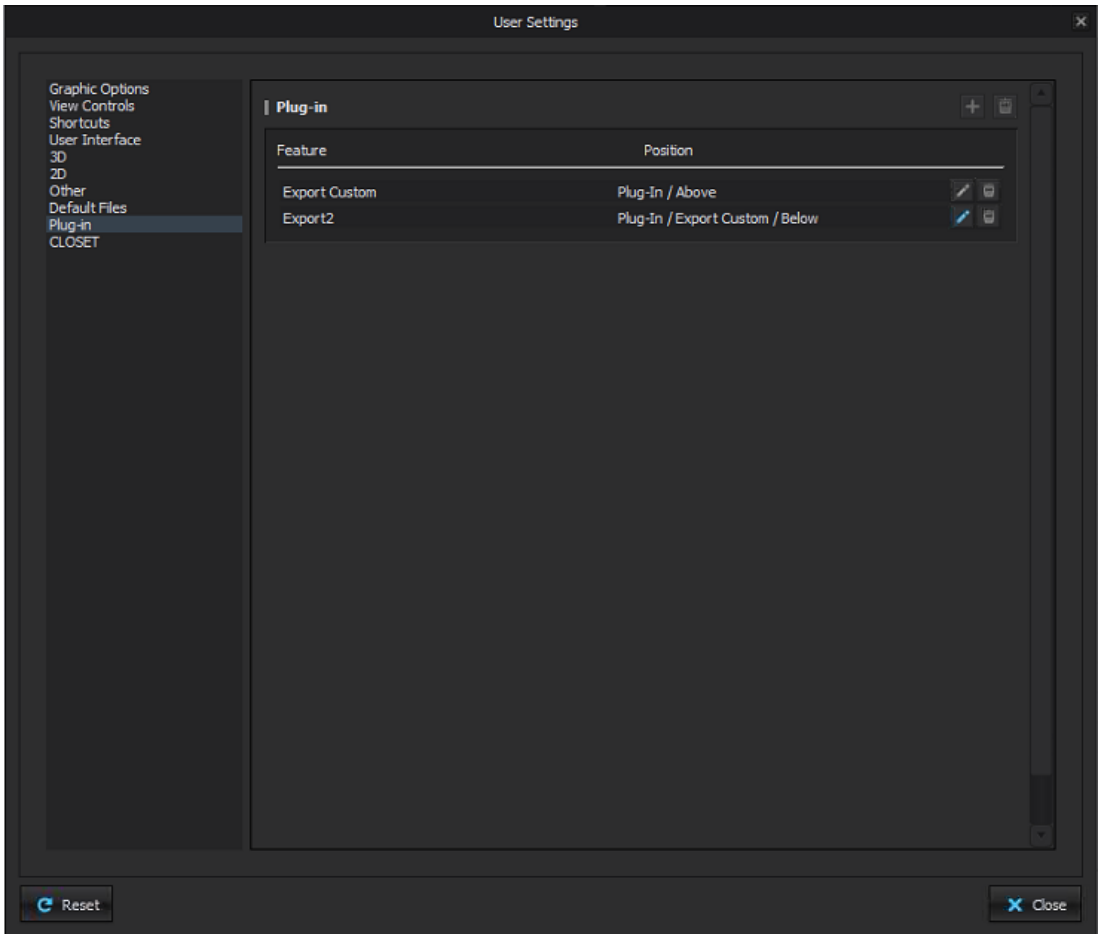
1) You can set the position to add a plug-in action into the desirable menu in the User Settings via User Settings -> Plug-in Tab.



2) If you click the ‘add’ icon, ‘File Open’ dialog will be shown to input the plug-in dll path for Windows, the dylib path for Mac OS and then ‘Register Plug-in’ dialog will appear; you can edit the feature title and menu position to insert the plug-in feature.



3) New item will be added into the Preference Plug-In tab like below. You can edit / delete the items.



※ In case you want to use the plug-in via the manager, please keep in mind you should not use the default plug-in because it can cause malfunctioning of CLO plug-in or the misconception of use.

7. Library Window Implementation

Library Window Interface are different from the export plug-in. When the user clicks ‘API tab’ in the Finder in CLO, the application starts to call the virtual functions in the Library Window API interface class. If you implemented a class inherited from the Library Window API interface class, build it, and put the dll into CLO’s executable folder as designated way; the module will run the implemented function inside the plug-in dll.

1) Windows

a. Open the sample project

Browse <api_sdk_package> - ‘CLOAPIInterface’ -> ‘Samples’ -> ‘LibraryWindowImplementation‘ and open the solution file: “LibraryWindowImplementation.sln”.

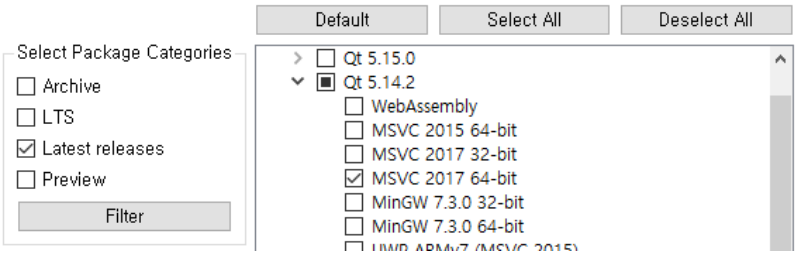
b. Write Code

You can write code as you want in LibraryImplePlugin class functions in .cpp file but do not add or modify anything inside .h file. Please see the **2) Mac OS in this chapter** for the codes.

c. Set Qt path

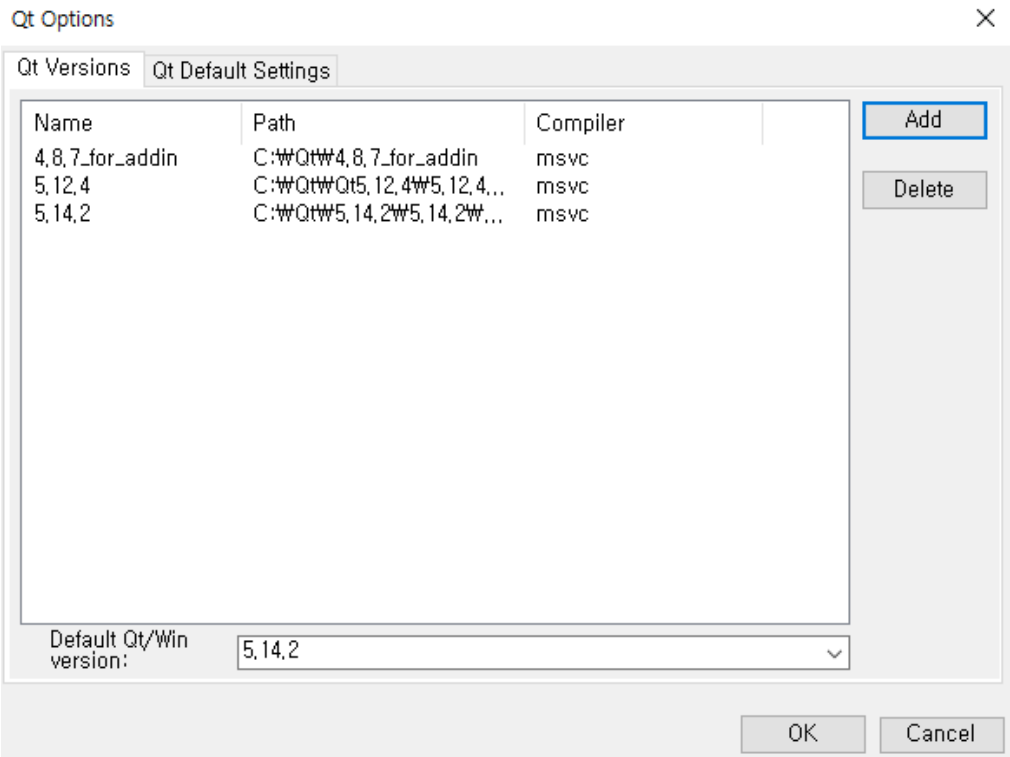
- Download Qt 5.14.2 library from https://www.qt.io/download-qt-installer

- Make sure to click “Deselect All” button when you come to select component window. Click Qt 5.14.2 and then open up the Qt 5.14.2 tab to click MSVC 2017 64-bit.



- In order to use the Qt, we need to install Qt VS Tools extension in Visual Studio. Click Tools -> Extensions and Updates then search for Qt Visual Studio Tools.

- Once Qt Visual Studio Tools is installed, Qt VS Tools tab will appear on the top of visual studio. Click Qt VS Tools -> Qt Options -> Add (search and add Path where qmake is located) -> OK -> default Qt/ Win version: 5.14.2 -> OK



d. Deploy Sample Assets
- Get the sample assets for the Library Window Implementation from https://s3.amazonaws.com/Outside_Work/CLO_API/2_7/sample_assets.zip or https://clo3d.oss-cn-shanghai.aliyuncs.com/web/CLO_API/2_7/sample_assets.zip
Extract the sample assets into 'C:/sample_assets'

e. Build the project

Build a solution/project then the output dll file will be created in LibraryWindowImplementation -> x64 ->Release folder named CloLibraryAPI_Plugin.dll. Copy the output dll file into the CLO executable folder. 'API tab' in Finder will act as you described in your code following the Library Window Interface/Implementation specification.

Please see the LibraryWindowInterfaceI.h and LibraryWindowimplmentation Plugin project code for details.

2) Mac OS

- This will break apple Code Signs and/or Apple Notarization
- This example assumes that the ID of the user logged in to mac is "current_user"

a. How to install Qt SDK

i. Download Qt mac sdk from https://www.qt.io/download-thank-you

ii. Install the downloaded Qt to the default folder

iii. Symbolic Link

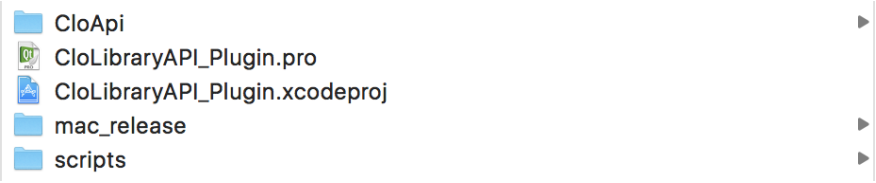
```
ln -s ~/(qt installed folder)/5.14.2/clang_64/ /usr/local/lib/QT
```

iv. Register QTDIR in .bash_profile.(If zsh, register.zsh_profile)

```
open ~/.bash_profile
export QTDIR=/usr/local/lib/QT
export PATH=$QTDIR/bin:$PATH
source ~/.bash_profile (or source .zsh_profile)
```

b. How to implement Library Window Implementation

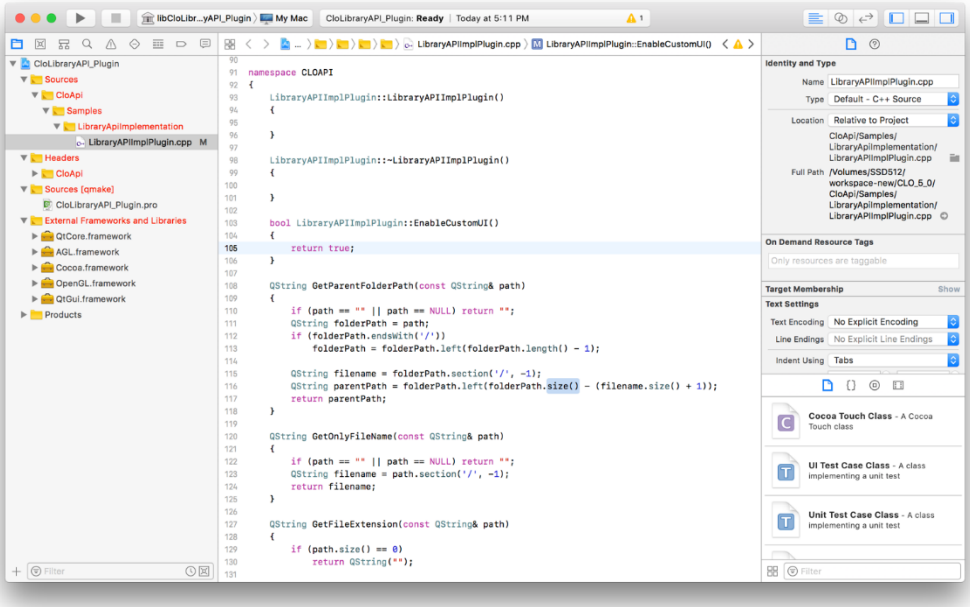
i. Extract the CLO_SDK.zip file.



ii. In the terminal, enter the following command to create CloLibraryAPI_Plugin.xcodeproj.

```
python scripts/build_libraryapi.py xdebug
```

iii. Open the generated xcodeproj file in Xcode.



iv. Edit the LibraryWindowImplPlugin.cpp file.
Always modify the next return value in the function to true.
The API tab is activated in the library window only if the return value of this function is true.
Please set USE_LIBRARY_WINDOW_CUSTOM_TAB to 1.

```
    bool LibraryWindowImplPlugin::EnableCustomUI()
    {
#if USE_LIBRARY_WINDOW_CUSTOM_TAB
        return true;
#else
        return false;
#endif
    }
```

v. You can set the tab name in Library Window

```
string LibraryWindowImplPlugin::GetTabName()
{
    string str = "Sample";
    return str;
}
```

vi. If you need to set the API tab as default and move to the first tab, set the ‘IsDefaultTab()’ as true. You can modify the macro, “USE_CUSTOM_TAB_AS_DEFAULT’ to 1 to set the API tab as default.

```
    bool LibraryWindowImplPlugin::IsDefaultTab()
    {
#if USE_CUSTOM_TAB_AS_DEFAULT
        return true;
#else
        return false;
#endif
    }
```

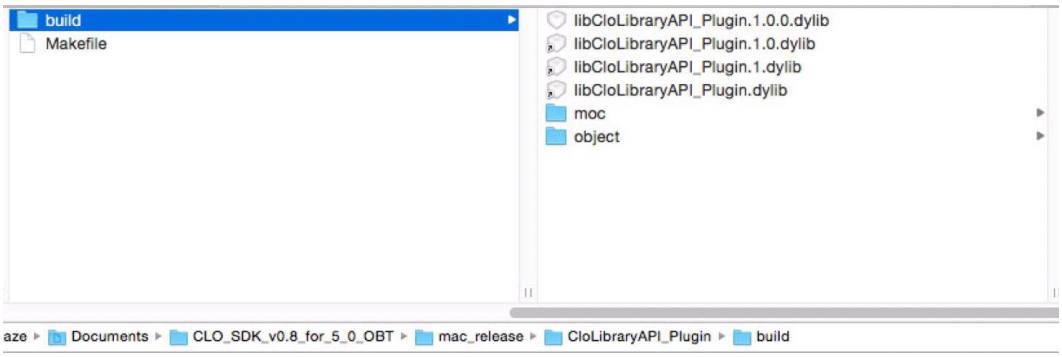
vii. The currently implemented code is a sample code.
Implement functions declared in LibraryWindowInterface.h to suit your requirements.

viii. Release build.

```
python scripts/build_libraryapi.py release
```

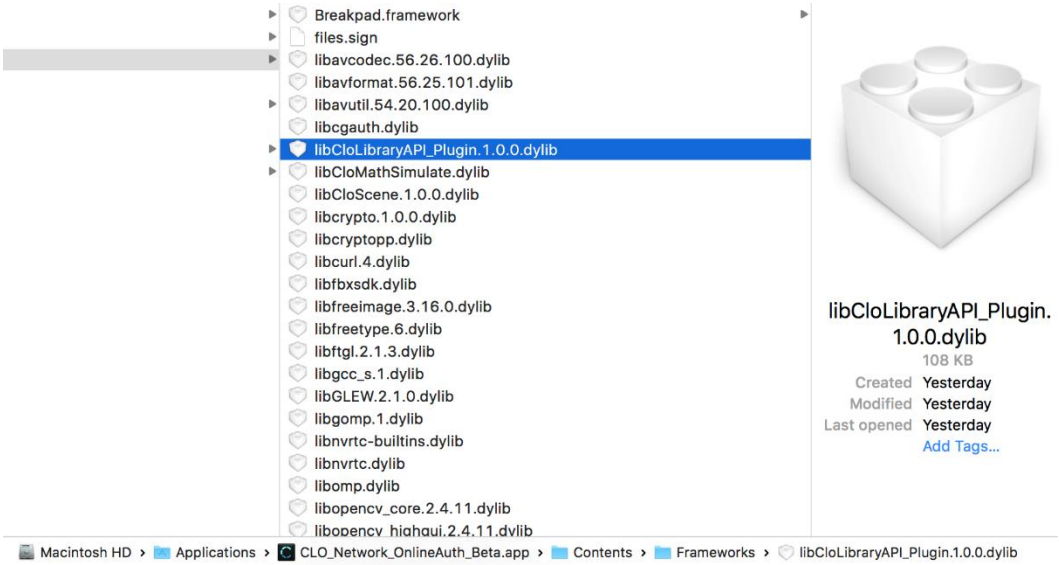
ix. The built dylib file will be created in the following location:

```
mac_release/CloLibraryAPI_Plugin/libCloLibraryAPI_Plugin.1.0.0.dylib
```



x. Overwrite the generated libCloLibraryAPI_Plugin.1.0.0.dylib file into the CLO package installed in /Applications:

/Applications/CLO_Network_OnlineAuth_Beta.app/Contents/Frameworks/libCloLibraryAPI_Plugin.1.0.0.dylib



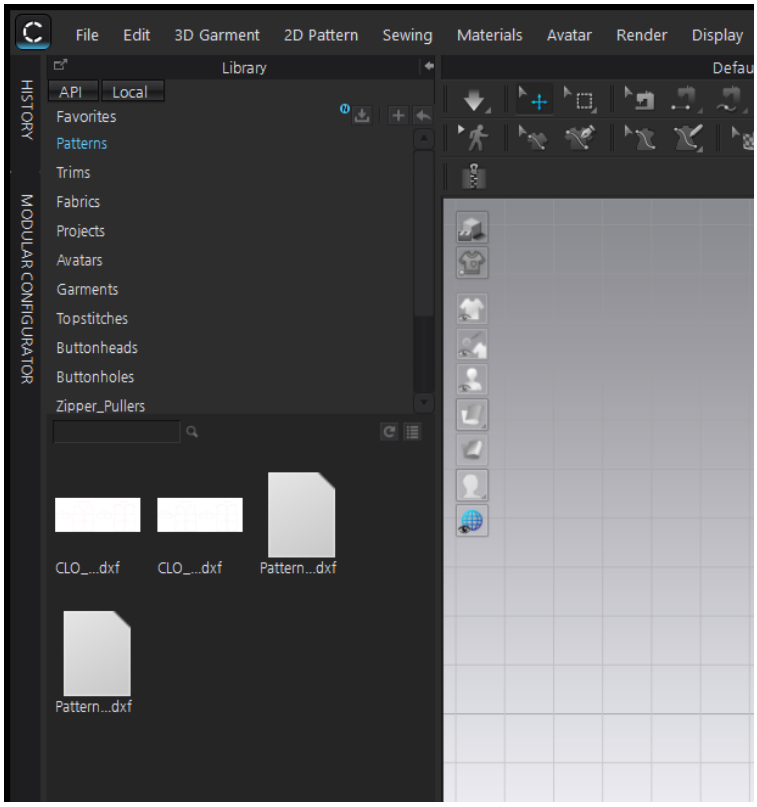
xi. Get the sample assets for the Library Window Implementation from https://s3.amazonaws.com/Outside_Work/CLO_API/2_7/sample_assets.zip or https://clo3d.oss-cn-shanghai.aliyuncs.com/web/CLO_API/2_7/sample_assets.zip

xii. Extract the sample assets into '/Users/current_user/sample_assets'.

xiii. Set the sample assets folder described in the 'APIDefine.h' to '/Users/current_user/sample_assets'

APIDefine.h ---------- const QString SAMPLE_STORAGE_DIRECTORY = QString("/Users/current_user/sample_assets/"

xiv. Now run CLO_Network_OnlineAuth.app.
Closet and API tabs will be created and shown on the left side of Favorites in the Library window.



xv. Click the API tab to navigate to the code you've implemented.

8. Plug-in Menu Position

1) “cloapi_plugins” folder

When CLO S/W starts to run, the plug-in module loads the dll(or dylib) files in the ‘cloapi_plugins’ folder automatically.

a. In case GetObjectNameTreeToAddAction() function is implemented and the default position for the plug-in menu is described inside the function properly, an action will be added to above or below the menu/action position for the plug-in action.

b. If GetObjectNameTreeToAddAction() is not implemented or contains improper description, an action will be added to the child of Settings / plug-in menu.

2) Edit

The user can modify the target position for the plug-in menu when or after adding the plug-in via the Plug-in manager. See “6. Plug-in manager”.

a. Add plug-in via Plug-in manager
If the plug-in dll(or dylib) has the proper GetObjectNameTreeToAddAction() function, the position will appear in the Register Plug-in dialog so that the user can edit and/or confirm.

b. Edit the plug-in action position
The user can change the action position via the plug-in manager for the plug-ins which were loaded automatically from ‘cloapi_plugins’ folder or were added from plug-in manager by user

※ Caution

Please keep in mind that the user edits the plug-in position via step 2), it would discard the default position described in the plug-in dll(or dylib) file. It means that the priority of the user modification is higher than the plug-in developers’; and the key value for the decision is the plug-in dll(or dylib) file path – absolute file path.
As this can make users confused as illustrated below, please be careful.

i. A developer wrote the default position inside ‘plug_in.dll’
ii. A user saves the ‘plug_in.dll’ into the ‘cloapi_plugins’ folder or a desirable folder, adds the plug-in and modifies the position
iii. The developer changes the default position inside ‘plug_in.dll’ and distributes it to the user again.

iv. If the user puts the latest ‘plug_in.dll’ into the same directory he or she used in ii) and overwrites it, the plug-in menu will be located in the position the user set in ii).

9. Plug-in Debug Logs

Plug-in log will be made in (CLO_ASSET_FOLDER)/api_plugin_log.txt when trying to run the plug-in action from the file menu.

1) CLO_ASSET_FOLDER

a. Windows

C:\Users\Public\Documents\CLO\Assets\

b. Mac CLO_ASSER_FOLDER

~/Documents/clo/Assets/

2) Debug logs

MESSAGE_PLUGIN_ACTION_MENU_CLICKED: "User clicked to run a plug-in action."

MESSAGE_PLUGIN_ACTION_TRYING_TO_FIND_REGISTERED_DLL_PATH: "Module starts to look up the registered plug-in file path for the plug-in action."

MESSAGE_PLUGIN_ACTION_SUCCESS_TO_FIND_REGISTERED_DLL_PATH: "Succeeded to look up the registered plug-in file path for the plug-in action."

MESSAGE_PLUGIN_ACTION_FAILURE_TO_FIND_REGISTERED_DLL_PATH: "Failed to look up the registered plug-in file path for the plug-in action."

MESSAGE_PLUGIN_ACTION_TRYING_TO_FIND_LOADED_PLUG_IN_DLL_FILE: "Checking if the plug-in file has been loaded or not."

MESSAGE_PLUGIN_ACTION_SUCCESS_TO_FIND_LOADED_PLUG_IN_DLL_FILE: "The plug-in has been loaded in the plug-in manager."

MESSAGE_PLUGIN_ACTION_TRYING_TO_LOAD_PLUG_IN_DLL_FILE: "Trying to load the plug-in file."

MESSAGE_PLUGIN_ACTION_SUCCESS_TO_LOAD_PLUG_IN_DLL_FILE: "The plug-in file is loaded into the plug-in manager successfully."

MESSAGE_PLUGIN_ACTION_FAILURE_TO_LOAD_PLUG_IN_DLL_FILE: "Failed to load the plug-in file."

MESSAGE_PLUGIN_ACTION_FAILURE_TO_LOAD_PLUG_IN_DLL_FILE_AND_ABORT: "Failed to load the plug-in file. Aborted."

MESSAGE_PLUGIN_ACTION_TRYING_TO_FIND_DO_FUNCTION: "Trying to find DoFunction inside the plug-in file."

MESSAGE_PLUGIN_ACTION_SUCCESS_TO_LOAD_DO_FUNCTION: "Succeeded."

MESSAGE_PLUGIN_ACTION_FAILURE_TO_LOAD_DO_FUNCTION: "Failed."

MESSAGE_PLUGIN_ACTION_TRYING_TO_EXECUTE_DO_FUNCTION: "Trying to run the DoFunction inside the plug-in file."

MESSAGE_PLUGIN_ACTION_SUCCESS_TO_EXECUTE_DO_FUNCTION = "Succeeded."

MESSAGE_PLUGIN_ACTION_EXEPCTION_TO_EXECUTE_DO_FUNCTION = "Exception."